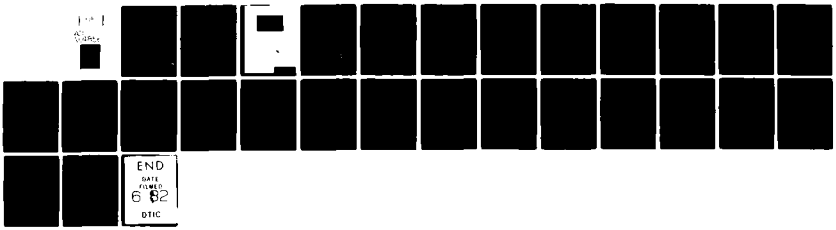
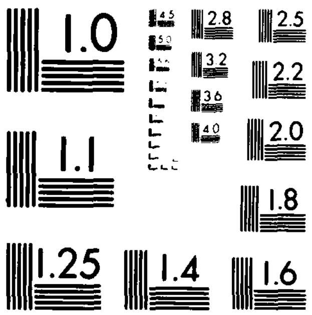

END
DATE FILMED
6 82
DTIC


1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A114856	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Real Time Resource Allocation in a Distributed System		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-06-82
7. AUTHOR(s) John Reif, Paul Spirakis		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0647
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Cambridge, MA 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, VA 22217		12. REPORT DATE February, 1982
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same as above		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

unlimited

This document has been approved for public release and sale; its distribution is unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

resource allocation, handshake communication, real-time algorithms, probabilistic algorithms, synchronization, dining philosophers, scheduling, two-phase locking.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see reverse side

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

82 05 26 042

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

In this paper we consider a resource allocation problem which is local in the sense that the number of users competing for a particular resource at any time instant is bounded and also at any time instant the number of resources that a user is willing to get is bounded. The problem may be viewed as distributedly achieving *matchings in dynamically changing hypergraphs*. We show that this problem is related to the fundamental problem of *handshake communication* (this problem can be viewed as achieving *matchings in dynamically changing graphs*, via distributed algorithms) in that an efficient solution to each of them implies an efficient solution to the other. We provide *real-time* solutions to the resource allocation problem (i.e., distributed algorithms with real time response) via probabilistic techniques. No probability assumptions about the system behavior are made, but processes are allowed the ability to make independent probabilistic choices. One of our solutions assumes the existence of an underlying efficient handshake communication system. Another is based on basic synchronization primitives (flag variables). The special case of equi-speed processes is examined. Applications are drawn to dining philosophers, scheduling and two-phase locking in databases.

REAL TIME RESOURCE ALLOCATION IN A DISTRIBUTED SYSTEM

John Reif
Paul Spirakis

TR-06-82

To appear in ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada, August 1982.

February 1982

REAL TIME RESOURCE ALLOCATION IN A DISTRIBUTED SYSTEM*

by

John Reif and Paul Spirakis

Aiken Computation Lab.

Harvard University

February 1982

*This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024, and the Office of Naval Research Contract N00014-80-C-0647.

ABSTRACT

is considered

In this paper we consider a resource allocation problem which is local in the sense that the number of users competing for a particular resource at any time instant is bounded and also at any time instant the number of resources that a user is willing to get is bounded. The problem may be viewed as distributedly achieving *matchings in dynamically changing hyper-graphs*. We show that this problem is related to the fundamental problem of *handshake communication* (this problem can be viewed as achieving *matchings in dynamically changing graphs*, via distributed algorithms) in that an efficient solution to each of them implies an efficient solution to the other. We provide *real-time* solutions to the resource allocation problem (i.e., distributed algorithms with real time response) via probabilistic techniques. No probability assumptions about the system behavior are made, but processes are allowed the ability to make independent probabilistic choices. One of our solutions assumes the existence of an underlying efficient handshake communication system. Another is based on basic synchronization primitives (flag variables). The special case of equi-speed processes is examined. Applications are drawn to dining philosophers, scheduling and two-phase locking in databases.

1. INTRODUCTION

1.1 The Resource Granting System

In this paper we consider a resource allocation problem which is *local* in the sense described in [Lynch, 1980]. The set of resources ρ and the set of user processes U may be infinite sets. However, there is a limit to the number of user processes requesting a particular resource. Resources are controlled by a set of *granting processes* R. Each granting process $j \in R$ controls a resource $\rho(j) \in \rho$. We assume user processes communicate only with those granting processes for which they request resources. (It is easy to superimpose each granting process into a requesting user process so that $U = R$).

A system described as above, is called a *Resource Granting System* (RGS). The goal of a RGS is to satisfy dynamically changing user requests for resource allocation. This is done in a distributed way, by only a local communication between *granting* and *requesting* processes. An *implementation* of the RGS determines the programs that the processes run. It is called *symmetric* if the programs do not depend on the location of the processes in the network. At each time $t \geq 0$ the actions of the requesting user processes U are specified by an adverse *oracle* $\mathcal{A}$, (which may be an "enemy" of the resource allocation algorithm, by setting actions in the worst way to increase the response time.) $\mathcal{A}$ also has the capability to select at time $t = 0$ the schedule of the speeds of all processes at all times $t \geq 0$. The oracle $\mathcal{A}$ is restricted, to allow users to keep asking for their resources for at least some nonzero length time interval. We assume that there is a global time t totally ordering events, but processes do not have access to it. *An RGS with priorities* is defined to be a resource granting system in which the requesting processes communicate to each resource granting process a rational number on the interval [0,1], indicating the priority of the request. These priorities can change dynamically, however they are assumed to preserve their value for at least a fixed constant number of process steps. The processes of R use these assigned priorities to grant their controlled resource with preference to users of higher priority. This must be done in a way avoiding user starvation.

1.2 Complexity of an RGS

In the sequel we consider a process to be *tame* during a time interval if it is speed bounded by fixed constants during that interval. We do not assume processes always tame, however they are supposed to be tame in the complexity analysis of response time. We require that, at no time, any granting process $i \in R$ simultaneously grant the resource $R(i)$ to more than one requesting process. We also require that, as soon as a process $j \in U$ has got all its required resources, then it can keep them only for a bounded length time interval (resulting in a bounded number of its steps, if the process is tame). Let resources(i) be the set of all possible resources that a process $i \in U$ is going to ever request, and let $\text{resources}_t(i)$ be the set of resources i is requesting at time instant t. Let $k_{i,t}$ be $|\text{resources}_t(i)|$. For each time t, the *willingness* digraph G_t is defined such that if $i \in U$ and $j \in R$ and if i requests (or has been granted) resource $\rho(j)$ at t, then the edge $i \underset{t}{\rightarrow} j$ belongs to G_t. Let also $j \underset{t}{\rightarrow} i$ if the granting process j is willing to allocate (or has granted) its resource to user i. Let v_t be the maximum valence of the nodes j of G_t such that $j \in R$, at time t. In the following we will assume that v_t and $k_{i,t}$ are above bounded by constants v,k at any time t and that $v \leq k$. This does not imply anything about the sets resources(i) $\forall i \in U$, which could be unbounded. We also assume, as in [Lynch, 1980], that each resource allocator $j \in R$ has a set S_j available to it of size $\leq v$, containing the names of those processes willing to get the resource. We assume this to be a primitive of our system (which could be implemented by a queued message system or by other means). Finally, we restrict any oracle $\mathscr{A}$ so that as soon as it produces a request for a resource (i.e., it orders the appearance of the edge $i \underset{t}{\rightarrow} j$ in G_t for $i \in U$, $j \in R$) it has to insist on that request for at least a bounded time interval. Note that the relation $\underset{t}{\rightarrow}$ can be viewed as a time varying hypergraph with node set $\Pi = U \cup R$ and edge set $E = \{\{i\} \cup \text{resource}_t(i),\ i \in U\}$. An RGS implementation dynamically achieves matchings in this hypergraph. We allow *probabilistic* RGS implementations where processes can use independent random number generators.

Fix an RGS implementation which may be probabilistic. In the following we assume processes to be tame over the stated time intervals. For each k $(0 \leq k \leq v)$ and oracle $\mathscr{A}$ let the k-*grant response* be the random variable $\gamma_{k,\mathscr{A}}$ giving the length of the interval Δ required for any process $i \in U$ to have k resource requests simultaneously granted, given that i requested these resources during the entire interval Δ, with priority 1. Let the *mean* k-*grant response* be $\bar{\gamma}_k = \max\{\text{mean}\{\gamma_{k,\mathscr{A}}\}$ over all oracles $\mathscr{A}\}$. For each ε in $[0,1]$ let the ε-*error* k-*grant response* be the minimum $\gamma_k(\varepsilon)$ such that for every oracle

$$\text{Prob}\{\gamma_{k,\mathscr{A}} \leq \gamma_k(\varepsilon)\} \geq 1 - \varepsilon \quad .$$

The RGS implementation is *real time* if for every $k \in \{1,\ldots,v\}$ and for every $\varepsilon \in (0,1]$, $\gamma_k(\varepsilon) > 0$ and independent of any global measure of the network (except v). The network here has as nodes the elements of Π and its edges $\{u,r\}$ mean that $\rho(r) \in \text{resources}(u)$. Note that if the RGS implementation is real time, then $\bar{\gamma}_k$ is constant, independent of any global measure of the graph H of the network (except v).

1.3 Previous Work

[Lynch, 1980] considered the problem of fast allocation of resources in a distributed system. Her RGS implementation was a deterministic, non-symmetric one (processes were allowed to know the color of each resource in a coloration of the resource graph) and the communication system adopted was a *message* system requiring buffered communication. The k-grant response was of the order $x(H)\cdot v^{x(H)}\cdot\tau$ where $x(H)$ is the chromatic number of the resource graph and τ is the time required for process communication. Note that since resources(i) for $i\in U$ may be unbounded, in the worst case $x(H)$ can be as large as the number of processes $|\Pi|$.

1.4 Results of this Paper

We shall present in Section 3 a *probabilistic* implementation of an RGS which has mean k-grant response $\bar{\gamma}_k = O(kv^k\cdot\bar{\tau}\cdot\log v)$ and ε-error k-grant response

$$\gamma_k(\varepsilon) = O\left(kv^k\log\left(\frac{1}{\varepsilon}\right)\tau\left(\frac{\varepsilon}{2v}\right)\right)$$

where $\bar{\tau}$ and $\tau(\varepsilon)$ are the mean time and ε-error time required for *handshake communication* between processes (see Appendix I for definitions). In [Reif, Spirakis 1981], [Reif, Spirakis 1982] handshake communication implementations were given with $\bar{\tau} = O(v^2)$ and $\tau(\varepsilon) = O(v^2\log(1/\varepsilon))$. Note that these implementations achieve real time, thus the resulting RGS implementation is also real time with

$$\bar{\gamma}_k = O(kv^{k\;2}\log v) \quad\text{and}\quad \gamma_k(\varepsilon) = O\left(kv^{k+2}\log\left(\frac{v}{\varepsilon}\right)\log\left(\frac{1}{\varepsilon}\right)\right).$$

However, any other handshake communication implementation would also do.

In Section 4 we present a basic way to implement a real time RGS in both cases of synchronous and asynchronous processes. The implementation is probabilistic. No underlying handshake communication system is assumed. Instead, the means of synchronization between processes in U and R are *flag* variables (which are written by just one process and allowed to be read by at most one other process). The response time has mean $\bar{\gamma}_k = O(kv^{k+1})$ and $\gamma_k(\varepsilon) = O(kv^{k+1} \log(1/\varepsilon))$.

1.5 Organization of Paper

This paper is organized as follows: Section 2 contains applications of RGS to dining philosophers, scheduling, two-phase locking in databases, and real-time handshake communications. Section 3 discusses a real time RGS assuming an underlying real time handshake communication system. Section 4 discusses a real time RGS implementation by use of low level synchronization primitives (i.e. boolean and rational flags). Section 4 first discusses an implementation in which processes have the same speeds but their actions are relatively shifted in time. At the end of Section 4 we generalize our algorithms to the case where processes are tame. The Appendix defines handshake communication systems and their performance measures.

2. APPLICATIONS

2.1 Hasty Dining Philisophers

As a simple example, we consider an interesting RGS system which we call *"hasty dining philosophers"*. Let the requesting processes U be distinct integers, $r_0, \ldots, r_{n-1}$, and the granting processes R be $0, \ldots, n-1$ so that $U \cap R = \emptyset$. The resources are *forks* $\{\rho(0), \ldots, \rho(n-1)\}$. Each philosopher

$r_j \in U$ has resources(r_j) consisting of the forks $\{\rho(r_j), \rho(r_{(j+1) \bmod n})\}$. Thus, the resource graph H is a cycle of length n. The "hasty dining philosophers" has a high level real time RGS implementation with mean 2-grant response $\bar{\gamma}_2 = O(1)$ and ε-error 2-grant response $\gamma_2(\varepsilon) = O(\log(1/\varepsilon)^2)$. The low level RGS implementation gives $\bar{\gamma}_2 = O(1)$ and $\gamma_2(\varepsilon) = O(\log(1/\varepsilon))$. Intuitively, the RGS implementation requires each philosopher r_j to be at any time granted both forks of resource(r_j) in expected constant time, but r_j must be "hasty" and relinquish these resources within constant time interval. Note that for each $i \in \{0,\ldots,n-1\}$ the granting process i can be placed within process r_i, thus resulting in essentially only n processes.

2.2 Scheduling

Suppose an acyclic digraph D is given with node out-degree $\leq k$ and in-degree $\leq v$. This graph can be separated into levels. We assume processes residing in the nodes of D can operate only after getting all their *resources* (which reside at the nodes which have no successors). Each process operates just once and then becomes a resource granting process, to serve the next higher level. All its successors are deleted. So, each node is initially a requesting process and after it gets all its resources once, it becomes a resource granting process. By assuming a real time RGS implementation the above system can be processed in time of the order of the depth of the digraph.

2.3 Two Phase Locking in Databases

Two-phase locking is a concurrency control method in databases (see for example [Bernstein, Goodman 1980]) with the feature that as soon as a transaction releases a lock, it never obtains additional ones. The technique of two-phase locking produces serializable transaction executions. We propose here a way to implement two-phase locking in a distributed database, which is different from any known algorithms. The underlying assumption is that transactions are allowed to act on the data only if they got all the locks. Let the processes of the user set U be called *transaction modules* and the processes of the set R be called *data modules*. If each transaction requests to lock at most k data modules at a time and if at most v transactions can compete for a lock at a time instant t, then a real time RGS will result in real time lock allocation per transaction. The transaction modules go through a "round" (of constant number of steps in duration) during which they communicate with all data modules they want to lock. They keep the locks they get only for constant number of steps hoping in the meanwhile to get all of them. If the round finishes and they did not succeed in getting all the locks, then they release whatever they got and try again in the next round. Given the previously stated response time $\gamma_k(\varepsilon)$ for a real-time RGS, one can now decompose the distributed system into a system of parallel servers with geometrically distributed service time, (one per transaction module). It is straightforward then to analyze the transaction waiting time, throughput etc., by assuming a probability distribution in the transaction arrival rate.

2.4 Handshake Communication in Real Time Via a Real Time RGS

We can implement here handshake communication in the sense of the Appendix, assuming the existence of a real-time RGS. Assume that each of the processes $j \in R$ control just one resource called the *channel* and when they allocate this resource, we say they *open* the channel. Each of the processes

$i \in U$ have $|\text{resource}_t(i)| \leq 1$, so as soon as any process $i \in U$ is granted one resource, process i does not compete for any other, until i releases that resource. The processes $i \in U$ are assumed to open their channel when they are granted their resource and to close their channel when the resource is removed. Given bounds v on the processes $i \in U$ competing for the same $\rho(j)$, $j \in R$ and a bound $k \leq v$ on the number of resources a user is *willing* to be granted at any time t, a real-time RGS will imply a *real-time handshake communication scheduler*, with the same time performance (see Appendix I). It is interesting to note (as we show in Section 3) that one can implement also a real time RGS by a real time distributed communication subsystem.

3. HIGH LEVEL RGS IMPLEMENTATION ASSUMING A REAL-TIME HANDSHAKE COMMUNICATION SYSTEM

3.1 The Algorithm

We shall assume here the existence of a DCS [as in Appendix I]. Then, the implementation of a probabilistic real time RGS is as follows:

The granting processes i are always willing to communicate only to the requesting processes of the set S_i (as defined in the introduction). The requesting processes are willing to communicate only to those granting processes whose resources they want (or have been allocated). By communication here we mean a handshake communication.

The granting processes do forever the following *grant algorithm* which is a loop, a single execution of which is called a *grant-phase*:

<u>Grant Algorithm</u>

for process $i \in R$

```
Do forever
    begin
    [1]. Do a handshake communication with anyone of the requesting
         processes in $S_i$ and get their priorities.
    [2]. Probabilistically select $j \in S_i$. (The values of the priorities
         determine the probability of each requesting process to be
         selected as determined in Section 3.2.)
    [3]. On first handshake with the selected process j, say "yes" to j
         allocating your resource to j.
    [4]. For $2.\bar{s}$ steps, the granting process says "no" to any requesting
         process but j in any handshake and says "yes" to j on any
         communication.
         (Here $\bar{s}$ contains at least time $\tau(\varepsilon)$ and it must be exactly
         equal to $\tau(\varepsilon)$ in duration, if i goes as fast as it can. Assume
         that, for tame processes, the duration of a step is between two
         nonzero values, $\delta_{min}$, $\delta_{max}$. Then $\bar{s} = \tau(\varepsilon)/\delta_{min}$.)
    [5]. On handshake with any other process than j, the granting process
         i says "no". On first handshake communication with j, the
         granting process i says "no" to j, indicating that resource
         $\rho(i)$ has been withdrawn from j and ending the grant phase.
    [6]. Wait for w steps randomly chosen from $[0,6\bar{s}]$.
    end
```

The requesting processes continuously attempt to communicate with the resources they want to be granted.

We may view the actions of the requesting processes time-sliced in *rounds* each round being the minimal time interval in which i communicates at least once with all k of the resource controlling processes of the resources which i wishes to obtain.

3.2 Probabilistic Selection

We give here a very simple implementation of probabilistic selection of one out of $\leq v$ processes (using their priorities) in $O(v)$ steps as required in phase 2 of the algorithm in Section 3.1. This implementation can easily be improved to $O(\log v)$ steps (see [Reif, Spirakis 1982]).

Suppose that each resource allocator i has just a random number generator drawing uniform numbers between 0 and 1. Let $p_{i1}, p_{i2}, \ldots, p_{iv}$ be the priorities of the processes requesting the resource of i at the current time.

To implement the selection process we do the following:

[2.1]. draw a random number r in $[0,1]$.

[2.2]. find the process name x for which

$$\frac{\sum_{k=1}^{x-1} p_{ik}}{\sum_{k=1}^{v} p_{ik}} < r \leq \frac{\sum_{k=1}^{x} p_{ik}}{\sum_{k=1}^{v} p_{ik}} \quad .$$

Note that stage [2.1] takes one step and stage [2.2] takes $O(v)$ steps since we have to evaluate 2 partial sums each time and test. The current partial sums can be evaluated from previous partial sums by a single addition.

3.3 Analysis

We now probabilistically analyze the algorithm of Section 3.1. By the stated properties of DCS,

$$\text{Prob}\{\text{a round length is } \leq \tau(\varepsilon)\} \geq (1-\varepsilon)^v \quad .$$

It is easy to see that

LEMMA 3.1. *The average length in steps of a grant phase is* $\leq 12\,\bar{s}$.

Proof. The stages 1, 3 and 5 of the algorithm will take $\bar{s}$ steps each by properties of DCS. Stage 2 takes steps $\leq \bar{s} = O(v)$ for known implementations of DCS. Stage 3 is guaranteed to have $2\bar{s}$ steps. Stage 6 takes $\leq 6\bar{s}$ steps. From now on we will assume (for simplicity) that all priorities are equal to 1. Let X be the event "a requesting processes gets a "yes" answer (in a handshake in stage 3)". We have:

$$\begin{aligned} &\text{Prob}\{X\} \\ &\quad = \text{Prob}\{\text{it actually competes for the resource}\} \\ &\quad\quad \text{Prob}\{\text{it gets a "yes" given it actually competes}\} \\ &\quad \geq \frac{1}{12} \cdot \frac{1}{v} \quad . \end{aligned}$$

Note that the length of the granting phase of each resource allocator is chosen in such a way to allow a requesting process to have all the resources for at least one of its steps, given that it gets all of them in a round.

If we consider a subclass of oracles $\mathscr{C}$ which put maximum contention on the system, then these oracles will give the worst case of the response time. However, in this case

$$\text{Prob}\{X\} \leq \frac{1}{v} \;, \qquad \text{for oracles } \mathscr{A} \in \mathscr{C} \;.$$

Let u be the random variable giving the number of rounds required for process i to succeed in being granted all k resources in one round. Then

$$\text{prob}\{u=m\} \leq \left(1 - \frac{1}{(12v)^k}\right)^{m-1} \left(\frac{1}{v}\right)^k$$

implying $\qquad \text{mean}(u) \leq (12\ v)^k.$

Let $u(\varepsilon)$ be the minimum value such that

$$\text{prob}\{u > u(\varepsilon)\} \leqslant \varepsilon \,, \qquad \forall\varepsilon \text{ in } (0,1] \,.$$

By properties of tails of geometrics

$$u(\varepsilon) = \frac{\log\left(\frac{1}{(12v)^k}\varepsilon\right) - \log\left(\frac{1}{(v)^k}\right)}{\log\left(1 - \frac{1}{(12v)^k}\right)}$$

$$= k(12v)^k \log\left(\frac{12}{\varepsilon}\right)$$

$$= O\left(kv^k \log\left(\frac{1}{\varepsilon}\right)\right) \,, \quad \text{for large } v.$$

For $\varepsilon \in (0,1]$

$$\text{Prob}\left\{\gamma_{k,\mathscr{A}} \leqslant \tau\left(\frac{\varepsilon}{2v}\right) \; u\left(\frac{\varepsilon}{2}\right)\right\}$$

$$\geqslant \text{Prob}\left\{\text{each round} \leqslant \tau\left(\frac{\varepsilon}{2v}\right) \quad \text{and} \quad u \leqslant u\left(\frac{\varepsilon}{2}\right)\right\}$$

$$\geqslant \left(1 - \frac{\varepsilon}{2v}\right)^v \left(1 - \frac{\varepsilon}{2}\right) \geqslant \left(1 - \frac{\varepsilon}{2}\right)^2 \geqslant 1 - \varepsilon \,.$$

So

$$\text{Prob}\left\{\gamma_{k,\mathscr{A}} = O\left(v^k k \log\frac{1}{\varepsilon}\, \tau\left(\frac{\varepsilon}{2v}\right)\right)\right\} \geqslant 1 - \varepsilon$$

implying

$$\gamma_k(\varepsilon) = O\left(kv^k \log\left(\frac{1}{\varepsilon}\right) \tau\left(\frac{\varepsilon}{2v}\right)\right)$$

which gives

$$\bar{\gamma}_k = O(kv^k\, \bar{\tau} \log v)$$

<u>Note</u>: Using $\tau(\varepsilon) = O(v^2 \log(1/\varepsilon))$ and $\bar{\tau} = O(v^2)$ we get

$$\gamma_k(\varepsilon) = O\left(v^{k+2}k \log\left(\frac{1}{\varepsilon}\right) \log\left(\frac{v}{\varepsilon}\right)\right)$$

and

$$\bar{\gamma}_k = O(kv^{k+2} \log v) \quad ,$$

4. REAL TIME RGS IMPLEMENTATION BY USE OF FLAG VARIABLES

4.1 The Algorithm for the Case of Synchronous Processes

For simplicity, we shall temporarily assume here that all processes go by the same speed. (Section 4.3 drops the assumption of synchronous processes). However, we also allow for this case that for some time in the past this did not happen (the processes were asynchronous) and so, at $t > 0$ the execution of their programs in time may be shifted (in an adverse way) relative to each other.

The communication between granting and requesting processes is done here by flag variables. To read one flag requires one of the process steps. In case of priorities, some of the flags are allowed to have rational values between 0 and 1. The flags P_{ij} indicate the priority of user j with respect to resource i. In the simple case of equal priorities all flags are boolean. Each granting process i has for each requesting process j a special flag F_{ij} whose value indicates if the resource $p(i)$ is allocated to j. If j reads F_{ij} and finds it 0, then it understands that it lost the resource. The granting processes execute forever the following loop, called a *grant phase*:

Grant Algorithm

of Granting Process $i \in R$

```
Do forever
    begin
    [1]. Read the priority flags of the requesting processes in the set $S_i$.
    [2]. Probabilistically select each of the requesting processes $j \in S_i$
         according to their priorities (see Section 3.2).
    [3]. Set the flag $F_{ij}$ to 1 indicating that resource $\rho(i)$ has been
         allocated to process $j$.
    [4]. Sleep for $cv$ steps (i.e., do $cv$ no-ops).
    [5]. Set the *warning flag* $L_{ij}$ to indicate to $j$ that he will loose
         resource after at most $2cv$ steps. Wait $2cv$ steps.
    [6]. Set $F_{ij}$ to 0 (remove resource). Erase the warning by setting
         $L_{ij}$ to 0.
    [7]. Wait for $w$ steps where $w$ is a random integer selected uniformly
         from $[0,5cv]$.
    end
```

Note: Stages 1,2 each take $O(v)$ steps. The algorithm requires them to each take cv steps. This constant c is here a fixed constant, used in stages 4, 5,6.

Each user process $j \in U$ executes continously the following loop, a single execution of which is called a *round.*

```
Do forever
    begin
    [1]. Set $p_{ij}$ for each resource $\rho(i)$ requested by user $j$.
    [2]. Poll for $cv$ steps to see which resources have been awarded to
         user $j$. User $j$ considers the resource $\rho(i)$ *awarded* only if
         $\rho(i)$ has been both allocated $(F_{ij} = 1)$ and not yet warned $(L_{ij} = 0)$.
```

```
[3]. If all resources requested by  j  are awarded use them for
     $\mu << v$  steps  ($\mu$  is a constant, controlled by the implementation).

end
```

A *phase* of a grant process:

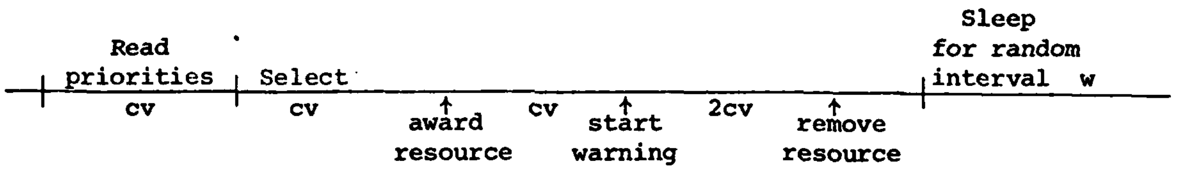


Figure 1

A *round* of a requesting process:

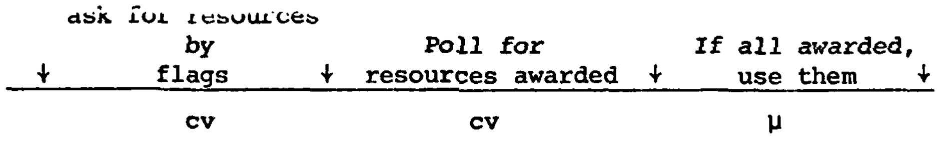


Figure 2

4.2 Analysis

Note that in the synchronous case assumed here, the time is essentially the processes steps. The power of the adversary is thus restricted to only a possibly malicious initial relative shift of the program counters of the various processes.

LEMMA 4.1. *It is impossible for the user j to conclude that it has got all resources and actually some of the resources to have been removed.*

Proof. Since the polling time of j lasts only cv steps, by the time he concludes that the last resource is allocated to him, the first allocated resource can at most be in the middle of the warning period (and hence not removed yet).

Note: The above lemma puts a limit on μ. It must be

$$\mu < cv - 1 \quad .$$

In the sequel we consider priorities $= 1$

LEMMA 4.2. *The probability that user* j *will get a particular resource in its current round is* $\geqslant 1/10v$.

Proof. This probability is equal to

Prob{flag of user j will be seen by the resource allocator in the current round}

$\cdot$ Prob{j will be selected given its flag was seen}

$$\geqslant \frac{1}{10} \cdot \frac{1}{v} \quad .$$ □

Note: The first probability is $\geqslant 1/10$ due to the random waits incurred by the resource allocators. These waits counteract the adverse relative shifts.

LEMMA 4.3. *The probability that user* j *will get a particular resource in its current round is* $\leqslant 1/v$ *for the worst case oracles.*

Proof. Consider oracles which put maximum contention in the system.

LEMMA 4.4. *The probability that user* j *will get all his resources in the same round is* $\geqslant 1/(10v)^k$ *and is* $\leqslant 1/v^k$.

Proof. By the fact that granting processes sleep for random intervals and hence their relative positions in the algorithm are statistically independent (and Lemmas 4.1, 4.3). □

Let $u = \#$ rounds required for user i to succeed in being granted all its k resources in one round. Then

$$\text{Prob}\{u = m\} \leq \left(1 - \frac{1}{(10v)^k}\right)^{m-1} \frac{1}{(v)^k}$$

implying

$$\text{mean}(u) \leq (10v)^k \ .$$

If $u(\varepsilon)$ is the least number such that $\text{prob}\{u > u(\varepsilon)\} \leq \varepsilon$ then

$$u(\varepsilon) = \frac{\log\left(\frac{\varepsilon}{(10v)^k}\right) - \log \frac{1}{v^k}}{\log\left(1 - \frac{1}{(10v)^k}\right)}$$

$$= k(10v)^k \log\left(\frac{1}{\varepsilon}\right) \qquad \text{for large } v$$

$$= O\left(kv^k \log\left(\frac{1}{\varepsilon}\right)\right) \ .$$

Since each round of i takes $< 3cv$ steps, we have

$$\text{Prob}\{\gamma_{k,\mathscr{A}} \leq 3cv\ u(\varepsilon)\} \geq 1 - \varepsilon$$

implying

$$\gamma_k(\varepsilon) = O\left(kv^{k+1} \log\left(\frac{1}{\varepsilon}\right)\right)$$

and

$$\bar{\gamma}_k = O(kv^{k+1}) \ .$$ □

Note that the RGS implementation by flags is more efficient than the RGS implementation by an underlying DCS system.

4.3 Asynchronous Case with Tame Processes

The algorithms and the analysis in this case are exactly the same with the synchronous case, with one change: The constant c must be replaced by $c' = c/\gamma_{min}$ in order to guarantee that $c'v$ steps of a process imply at least cv steps of any other process and at most $c'\gamma_{max}v$ steps. We again will get

$$\gamma_k(\varepsilon) = O\left(kv^{k+1}\log\left(\frac{1}{\varepsilon}\right)\right)$$

$$\bar{\gamma}_k = O(kv^{k+1}) \quad .$$

Note that these results are slightly better than those based on a handshake communication system, since there is no uncertainty about flag communication in each round. However, when processes are not tame, the *correctness* of the implementation by flags may be violated, while the correctness of the implementation by an underlying DCS will be preserved (because of the handshake communication which accompanies allocation or deallocation of a resource) given that the correctness of the underlying DCS is not violated when processes are not tame.

References

Andrews, G., "Synchronizing Resources," *ACM Transactions on Programming Languages and Systems*, Vol. 3, No. 4, Oct. 81, pp. 405-430.

Angluin, D., "Local and Global Properties in Networks of Processors," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency between Synchronous and Asynchronous Systems," *13th Annual Symposium on Theory of Computing*, April 1981.

Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes," *ACM Trans. on Prog. Lang. and Systems*, Vol. 2, No. 2, April 1980, pp. 234-238.

Bernstein, P. and N. Goodman, "Fundamental Algorithms for Concurrency Control in Distributed Database Systems," CCA TR. Contract No. F30603-79-0191, Cambridge, MA, 1980.

Dennis, J.B. and D.P. Misunas, "A Preliminary Architecture for a Basic Data-flow Processor," *Proc. of the 2nd Annual Symposium on Computer Architecture*, ACM, IEEE, 1974, pp. 126-132.

Fischer, M.J., N.A. Lynch, J.E. Burns, and A. Borodin, "Resource Allocation with Immunity to Limited Process Failure," 19th FOCS, 1979, pp. 234-254.

Francez, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 373-379.

Hoare, C.A.R., "Communicating Sequential Processes," *Com. of ACM*, Vol. 21, No. 8, Aug. 1978, pp. 666-677.

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in *8th ACM Symposium on Principles of Program Languages*, Jan. 1981.

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. Computer Science, Yale Univ., June 1977.

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium in Theory of Computing*, Los Angeles, California, April 1980, pp. 70-81.

Mahjoub, A., "Some Comments on Ada as a Real-time Programming Language," to appear.

Rabin, M., "N-Process Synchronization by a 4 $\log_2 N$-valued Shared Variable," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 407-410.

Rabin, M., "The Choice Coordination Problem," Mem. No. UCB/ERL M80/38, Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.

Reif, J.H., and Spirakis, P., "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time," *13th Annual ACM Symposium on Theory of Computation*, Wisconsin, 1981, pp. 133-145.

Reif, J.H., and Spirakis, P., "Unbounded Speed Variability in Distributed Communications Systems," *Ninth ACM Symposium on Principles of Programming Languages*, January 25-27, 1982, Albuquerque, New Mexico.

Schwarz, J., "Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburgh, 1980.

Silberschatz, A., "Extending CSP to Allow Dynamic Resource Allocation," Technical Report, Dept. of Computer Science, Univ. Texas, Austin, Texas, 1981.

Tonag, S., "Deadlock and Livelock-Free Packet Switching Networks," *12th Annual Sympsoim on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Valiant, L.G., "A Scheme for Fast Parallel Communication," Technical Report Computer Science Dept., Edinburgh, Scotland, July 1980.

APPENDIX I

Distributed (Handshake) Communication Systems (DCS)

Suppose that each process has a special resource called channel which can be in one of two states *open*, *closed*. A *handshake* of two processes i, j in time t is a combination of processes states at time t so that both channels of i and j are open at the same time.

We require that *successful direct communication* requires a handshake of at least 1 step overlap of both processes and that the handshake relation should be a matching. At any instant t no process is allowed to be handshaking with more than one other process. During the one step overlap, a message can be transmitted from one process to the other. The problem is usually to synchronize processes (via a distributed scheduler) so that they can handshake at their will, given that the means of synchronization is some low level construct (a message system, buffered communication, shared variables or flags) which does not guarantee the handshake property if used in an unsophisticated way. A distributed scheduler is called *real time* if it has the property that if two processes i,j are willing to handshake mutually for at least a constant time interval, then they will actually achieve successful direct communication during that time interval with arbitrarily small probability of error.

Formally, let $\tau(\varepsilon)$ be the smallest real number such that if two processes i,j are mutually willing to handshake for at least $\tau(\varepsilon)$ time, then they will actually succeed in 1 step overlap of open channels during that time, with probability $\geq 1-\varepsilon$. $\tau(\varepsilon)$ is called the *ε-error response* of the handshake algorithm. The *mean response*, $\bar{\tau}$ of a handshake algorithm is the maximum (over all adverse speed schedules of tame processes and overall adverse communication requests subject to restrictions stated in the introduction) of the mean time needed for two processes to handshake, from the time instant they start to be mutually willing. A real time probabilistic scheduler has $\tau(\varepsilon)$ depending only on v and not on any other global measure of the communications graph. (v is a fixed upper bound on the out-valence of the dynamic communication willingness digraph at any time instant t). We also require $\tau(\varepsilon)$ to increase at most linearly with $1/\varepsilon$. Note that such a scheduler has $\bar{\tau}$ also depending only on v.

The handshake problem has been given some attention in literature [Schwartz, 79], [Francez, Rodeh 80], [Francez, 81], [Reif, Spirakis 81], [Reif, Spirakis 82].

For Section 3 we require a *Distributed Communication System* (DCS) as defined above with a distributed real time probabilistic scheduler. We also require the DCS to have the following property:

If a process i is willing to communicate with $k \leqslant v$ processes for at least time $\geqslant \tau(\varepsilon)$ and if they are also willing to (handshake) communicate with i during that interval, then the probability that i will be able to communicate with all of them (in some order) within $\tau(\varepsilon)$, is $\geqslant (1-\varepsilon)^v$. Such a real time DCS was implemented in [Reif, Spirakis 81] with

$$\tau(\varepsilon) = O(v^2 \log(1/\varepsilon))$$

and

$$\bar{\tau} = O(v^2) .$$

DATE

ILME

—8